Лабораторная работа № 9

Модульное тестирование.

1. Цель работы.

Знакомство с xUnit. Использование xUnit для тестирования классов контроллера.

2. Общие сведения.

Примеры модульного тестирования с помощью xUnit см. здесь:

https://www.c-sharpcorner.com/article/unit-testing-with-xunit-and-moq-in-asp-net-core/

https://metanit.com/sharp/aspnet5/22.2.php

https://learn.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-with-dotnet-test

https://medium.com/bina-nusantara-it-division/a-comprehensive-guide-to-implementing-xunit-tests-in-c-net-b2eea43b48b

Описание библиотеки NSubstitute здесь:

https://nsubstitute.github.io/help/getting-started/

Для проверки результата выполнения теста xUnit предоставляет класс Assert, который содержит статические методы – утверждения:

- **Contains** – проверяет, что строка содержит указанную подстроку или коллекция содержит указанный элемент;
- **DoesNotContain** – проверяет, что строка НЕ содержит указанную подстроку или коллекция НЕ содержит указанный элемент;
- **DoesNotThrow** – проверяет, что код НЕ генерирует исключение;
- **Equal** – проверяет, что два объекта эквивалентны;
- **False** – проверяет условие на false;
- **InRange** – проверяет, что величина находится в указанных границах;

- **IsAssignableFrom** – проверяет, что объект принадлежит указанному типу или наследуется от него;
- **IsNotType** – проверяет, что объект НЕ принадлежит указанному типу;
- **IsType** – проверяет, что объект принадлежит указанному типу;
- **NotEmpty** – Проверяет, что коллекция содержит элементы;
- **NotEqual** – проверяет, что два объекта НЕ эквивалентны;
- **NotInRange** – проверяет, что величина находится ВНЕ указанных границ;
- **NotNull** – проверяет, что объект не Null;
- **NotSame** – проверяет, что два объекта НЕ представляют собой одну и ту же сущность;
- **Null** – проверяет, что объект равен Null;
- **Same** – проверяет, что два объекта представляют собой одну и ту же сущность;
- **Throws** – проверяет, что код генерирует исключение;
- **True** – проверяет условие на true.

Пример проверки на генерирование исключения:

```
[Fact]
public void StackLimitIsControlled()
{
      //arrange
      var stack = new StackDemo();
      stack.StackLimit = 0;
      //act
      Action testCode = () => stack.Push("test");
      //assert
      Assert.Throws<StackDemo.StackFullException>(testCode);
        }
```

xUnit поддерживает два разных типа модульных тестов - Fact и Theory. Тип теста указывается в виде атрибута.

Fact используется, когда у нас есть некоторые критерии, которые всегда должны выполняться независимо от данных.

Theory зависит от множества параметров и ее данных. Тест пройдет для определенного набора данных, а не других. Для передачи набора исходных данных используются атрибуты [InlineData], [ClassData], [MemberData]

InlineData - позволяет передавать простые объекты:

```
[Theory]
 [InlineData(2, 4, .5)]
 [InlineData(2, 4, 1)]
 [InlineData(3, 4, (double)3/4)]
public void CanDivide(int a, int b, double c)
{

}
```

ClassData – атрибут, позволяющий в качестве источника набора данных использовать класс. Класс, используемый в качестве источника данных, должен наследоваться от `IEnumerable<object[]>.`

Пример использования:

```
[Theory]
[ClassData(typeof(DataSource))]
public void CanDivide(int a, int b, double c)
{

}
```

MemberData - атрибут позволяющий в качестве источника тестовых данных использовать метод. Метод должен возвращать `IEnumerable<object[]>.`

Пример использования:

```
[Theory]
[MemberData(nameof(DataSource.GetTestData),
                MemberType =typeof(DataSource))]
public void CanDivide(int a, int b, double c)
```

```
{

}
```

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №8.

3.2. Подготовка проекта

Добавьте в решение проект модульных тестов xUnit. Имя проекта – XXX.Tests.

Добавьте в проект пакеты NuGet:

- NSubstitute – для имитации классов и интерфейсов
- Microsoft.AspNetCore.Mvc – для тестирования контроллеров
- Microsoft.EntityFrameworkCore.InMemory или Microsoft.EntityFrameworkCore.Sqlite – для имитации работы с контекстом базы данных

Добавьте ссылки (dependencies) на основной проект и на проекты XXX.Api, XXX.Domain

3.3. Задание №1. Тестирование контроллера Product

Разработайте модульные тесты для метода Index контроллера Product

3.3.1. Рекомендации к заданию 1

Код метода Index должен быть примерно следующим:

```
[Route("menu/{category?}")]
public async Task<IActionResult> Index(string? category, int pageNo=1)
{
        // получить список категорий
        var categoriesResponse = await _categoryService.GetCategoryListAsync();

        // если список не получен, вернуть код 404
        if(!categoriesResponse.Successfull)
                    return NotFound(categoriesResponse.ErrorMessage);

        // передать список категорий во ViewData
        ViewData["categories"]= categoriesResponse.Data;

        // передать во ViewData имя текущей категории
```

```
        var currentCategory = category==null
            ?"Все"
            : categoriesResponse
                        .Data
                        .FirstOrDefault(c=>c.NormalizedName==category)?.Name;
        ViewData["currentCategory"]= currentCategory;

        // получить список объектов
        var productResponse = await _service.GetProductListAsync(category, pageNo);

        // если список не получен, вернуть код 404
        if (!productResponse.Successfull)
            return NotFound(productResponse.ErrorMessage);

        // проверить, был ли запрос асинхронным
        if (Request.IsAjaxRequest())
            // в случае асинхронного запроса вернуть частичное представление
            return PartialView("_ListPartial", productResponse.Data);
        // вернуть полное представление
        return View(productResponse.Data);
}
```

Необходимо проверить:

- метод возвращает код 404 при неуспешном получении списка категорий
- метод возвращает код 404 при неуспешном получении списка объектов

При успешном получении списков проверить:

- во ViewData передан список категорий
- во ViewData передано правильное значение текущей категории («Все», если категория не указана, или переданное в метод имя категории)
- в представление передана модель – список объектов

Примечание: для асинхронных запросов тест можно не делать (не проверять, что передается частичное представление)

3.3.2. Рекомендации к заданию 1

В конструктор контроллера внедряются сервисы ICategoryService и IProductService. Для имитации работы тих сервисов используйте библиотеку NSubstitute.

Не нужно имитировать все методы интерфейсов. Достаточно имитировать только те методы, которые использует тестируемый метод контроллера (`GetProductListAsync и GetCategoryListAsync`)

Для имитации свойства Request – см. пример в презентации лекции.

3.4. Заданиие № 2. Тестирование сервиса ProductService проекта XXX.Api

Разработайте модульные тесты для метода `GetProductListAsync` класса `ProductService`.

Тесты должны проверить следующее:

- метод по умолчанию возвращает первую страницу размером 3 объекта на странице и правильно рассчитывает количество страниц
- метод правильно выбирает заданную страницу
- метод правильно выполняет фильтрацию объектов по категории
- метод не позволяет задать размер страницы больше максимального (см Лабораторную работу №4, п.3.7.)
- метод возвращает значение Success=false, если номер страницы превышает общее количество страниц.

3.4.1. Рекомендации к заданию №2

В методе `GetProductListAsync` объекты IWebHostEnvironment и IHttpContextAccessor не используются. Поэтому при создании объекта класса `ProductService` в конструктор можно передать «null».

Для имитации работы с базой данных при создании контекста базы данных используйте **In-memory провайдер** (не рекомендуется – см. https://learn.microsoft.com/en-us/ef/core/testing/testing-without-the-database#in-memory-provider) или **SQLite in-memory** (предпочтительно - см. https://learn.microsoft.com/en-us/ef/core/testing/testing-without-the-database#sqlite-in-memory).

Пример использования SQLite in-memory см. здесь: https://github.com/dotnet/EntityFramework.Docs/blob/main/samples/core/Testing/TestingWithoutTheDatabase/SqliteInMemoryBloggingControllerTest.cs

Пример теста того, что метод по умолчанию возвращает первую страницу размером 3 объекта на странице и правильно рассчитывает количество страниц:

```
[Fact]
public void ServiceReturnsFirstPageOfThreeItems()
{
      using var context = CreateContext();
      var service = new ProductServise(context, null, null);

      var result = service.GetProductListAsync(null).Result;

      Assert.IsType<ResponseData<ListModel<Dish>>>(result);
      Assert.True(result.Successfull);
      Assert.Equal(1, result.Data.CurrentPage);
      Assert.Equal(3, result.Data.Items.Count);
      Assert.Equal(2, result.Data.TotalPages);
      Assert.Equal(context.Dishes.First(), result.Data.Items[0]);
}
```

4. Контрольные вопросы

1. Для чего используют модульные тесты?
2. Что означает шаблон AAA в модульном тестировании?
3. Для чего используется библиотека NSubstitute?
4. Как в XUnit проверить, что значение типа int, возвращенное тестируемым методом, верно?
5. Как протестировать значение, передаваемое во ViewData?
6. Как в XUnit проверить, что метод генерирует исключение?
7. Как в NSubstitute создать имитацию метода класса?